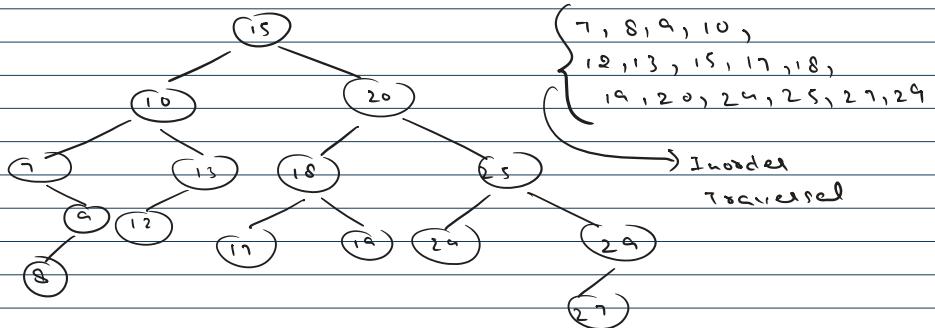
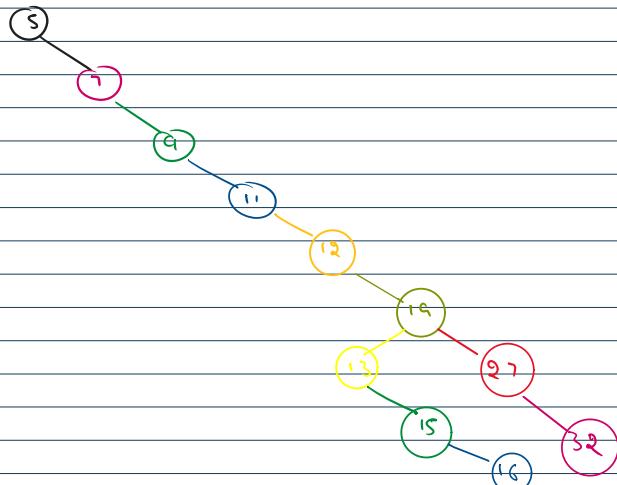


Binary Search Tree

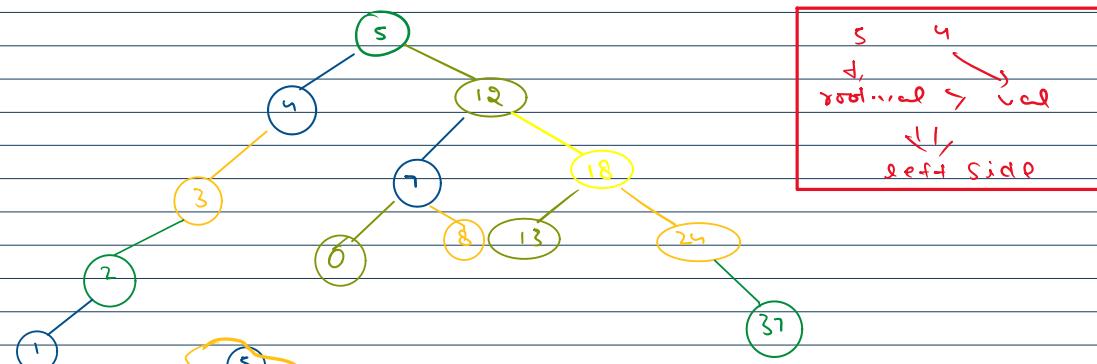
1) BST

2) left subtree $< \text{root}$ 3) right subtree $> \text{root}$ How to check if it is BSTBinary Search Tree Construction

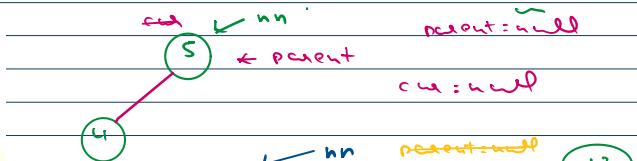
5, 7, 9, 11, 12, 19, 13, 27, 32, 15, 16



5, 4, 3, 19, 18, 2, 1, 24, 13, 37, 7, 8, 6



```
private Node insertIter(Node nn, int val) {
    Node node = new Node();
    node.val = val;
    if(nn == null) {
        return node;
    }
}
```

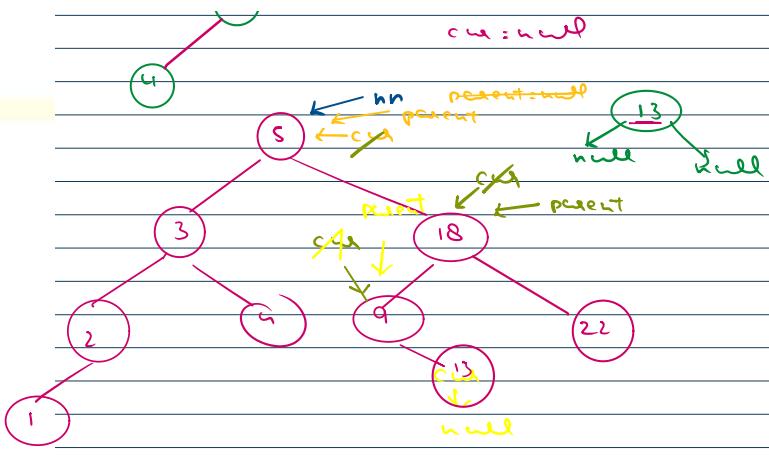


5, 4, 7, 2, 1, 8, 13, 6, 3

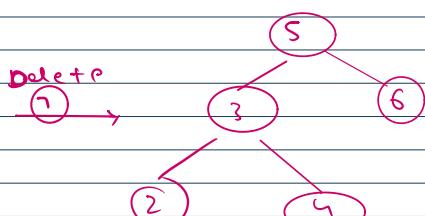
```

node.val = val;
if(nn == null) {
    return node;
}
Node cur = nn;
Node parent = null;
while(cur != null) {
    parent = cur;
    if(val < cur.val) {
        cur = cur.left;
    } else {
        cur = cur.right;
    }
}
if( val < parent.val) {
    parent.left = node;
} else {
    parent.right = node;
}
return nn;
}

```



Delete Node in BST

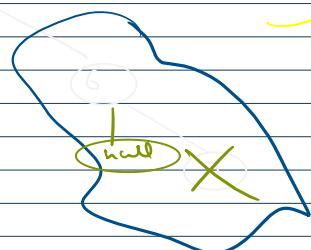


~~110 root~~

~~root~~

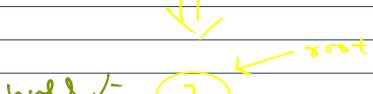
~~root.val > val \Rightarrow left subtree~~

~~root.val < val \Rightarrow right subtree~~

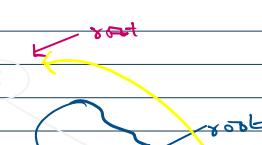
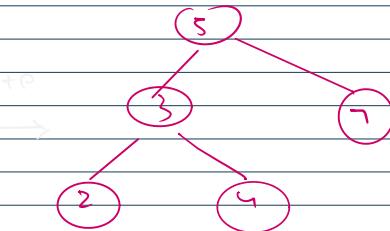


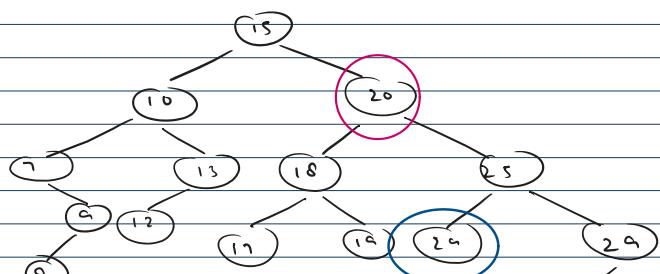
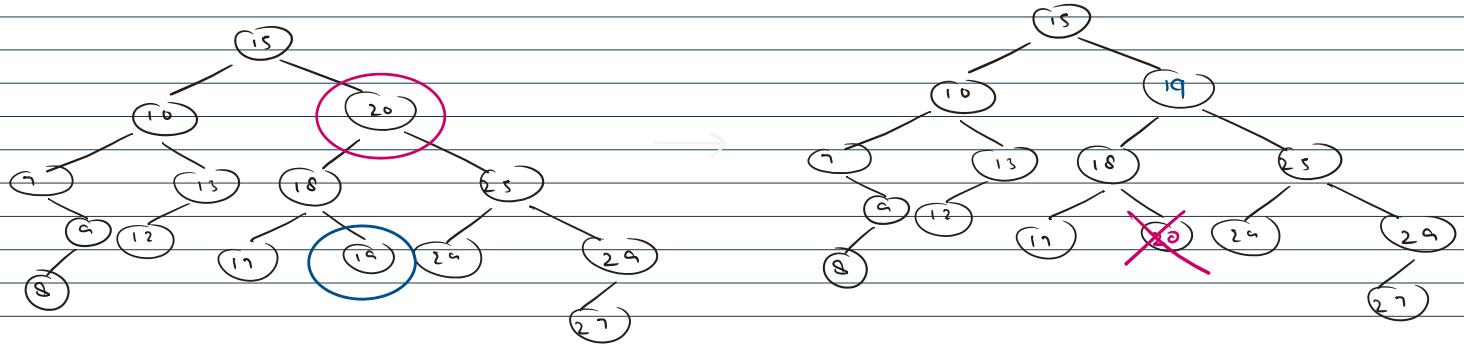
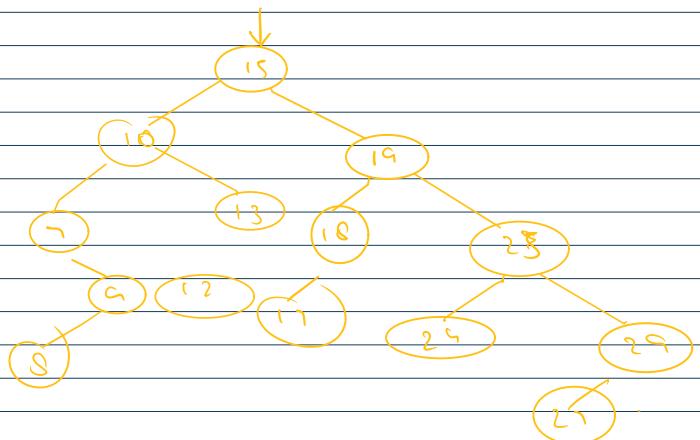
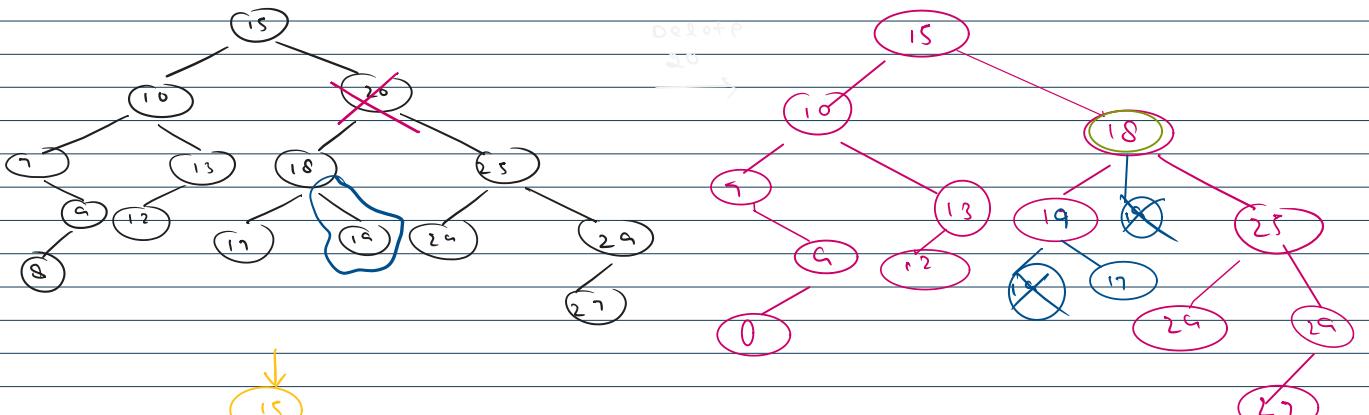
~~root~~

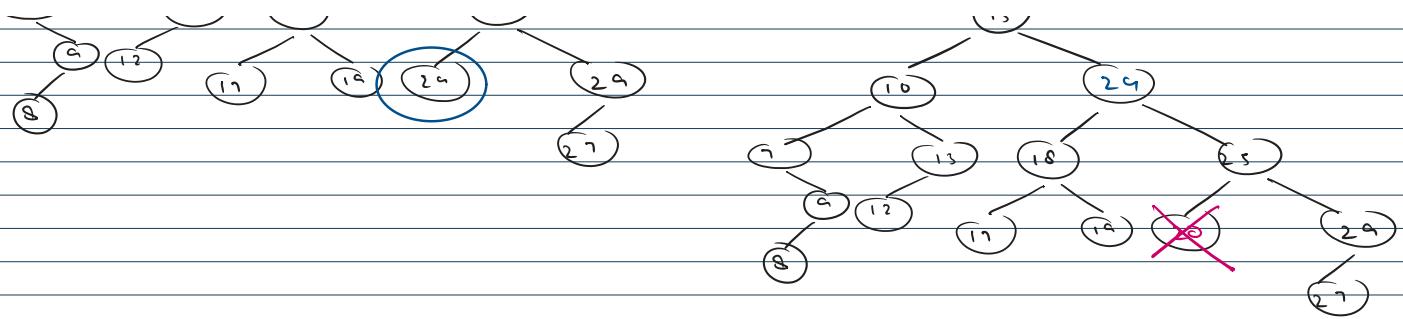
~~null \leftarrow 7~~



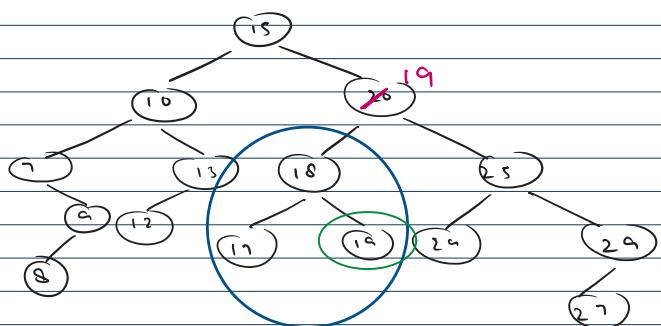
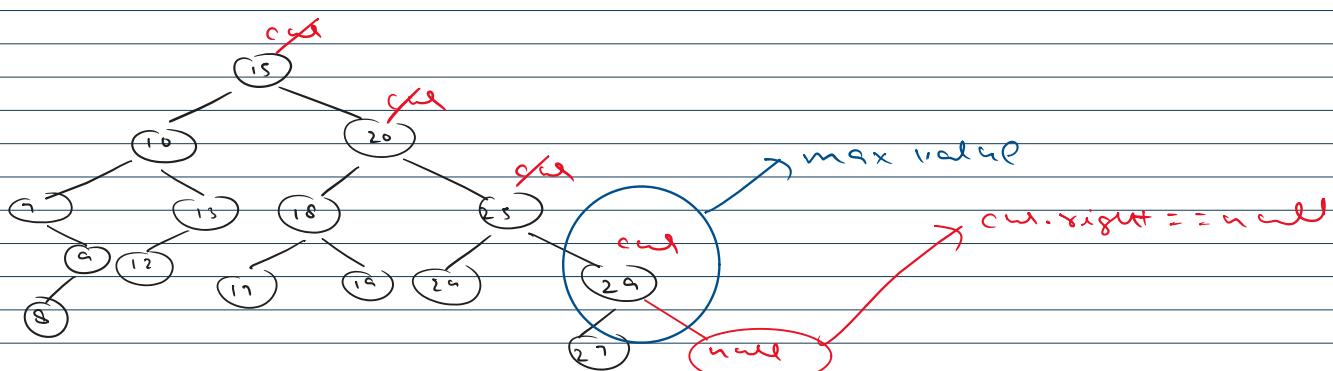
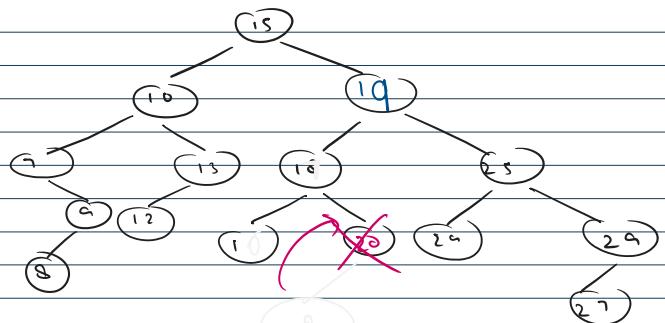
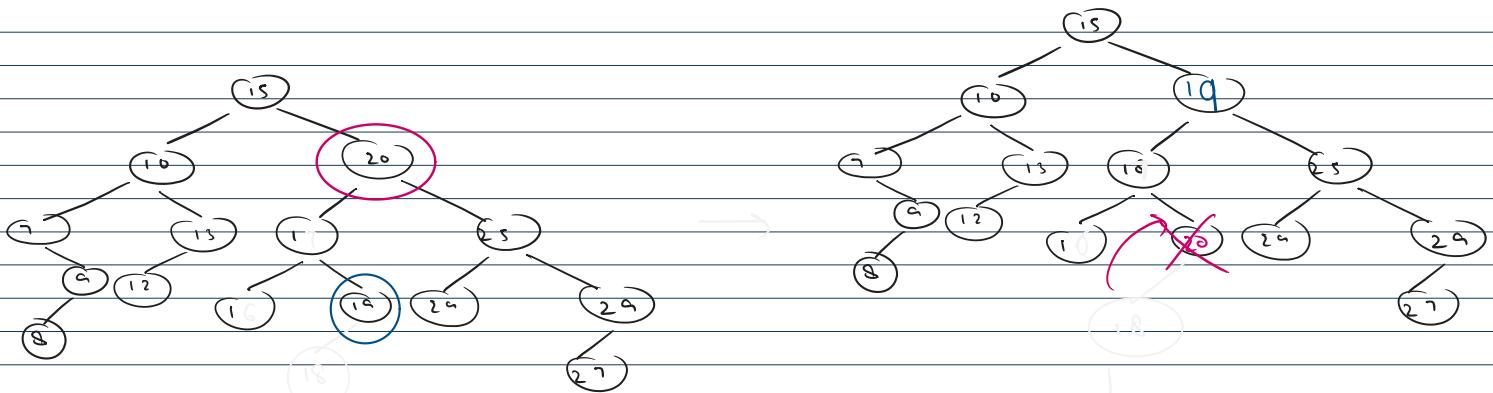
~~root~~

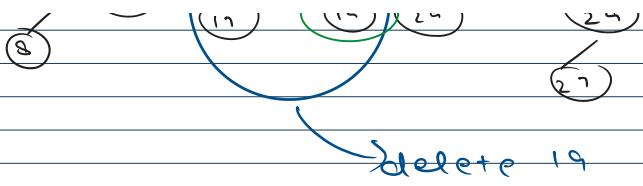






left subtree \rightarrow largest value ρ / \rightarrow swap
 right subtree \rightarrow smallest value ρ \





```

public TreeNode deleteNode(TreeNode root, int key) {
    if(root == null) {
        return null;
    }
    if(root.val < key) {
        root.right = deleteNode(root.right, key);
    }
    else if (root.val > key) {
        root.left = deleteNode(root.left, key);
    }
    else {
        if(root.left == null & root.right == null) {
            return null;
        }
        else if( root.left == null) {
            return root.right;
        }
        else if(root.right == null) {
            return root.left;
        }
        else {
            int max = max(root.left);
            root.val = max;
            root.left = deleteNode(root.left, max);
        }
    }
    return root;
}

```

The code implements a recursive deletion of a node with value 'key' from a binary search tree. It handles four cases for the target node: 1) it's null, 2) its value is less than 'key', 3) its value is greater than 'key', and 4) its value is equal to 'key'. In the fourth case, it checks if the node has left or right children. If neither exists, it returns null. If only the left child exists, it returns the right child. If only the right child exists, it returns the left child. Otherwise, it finds the maximum value in the left subtree (using a helper function 'max'), replaces the current node's value with this maximum, and then recursively deletes the maximum value from the left subtree.

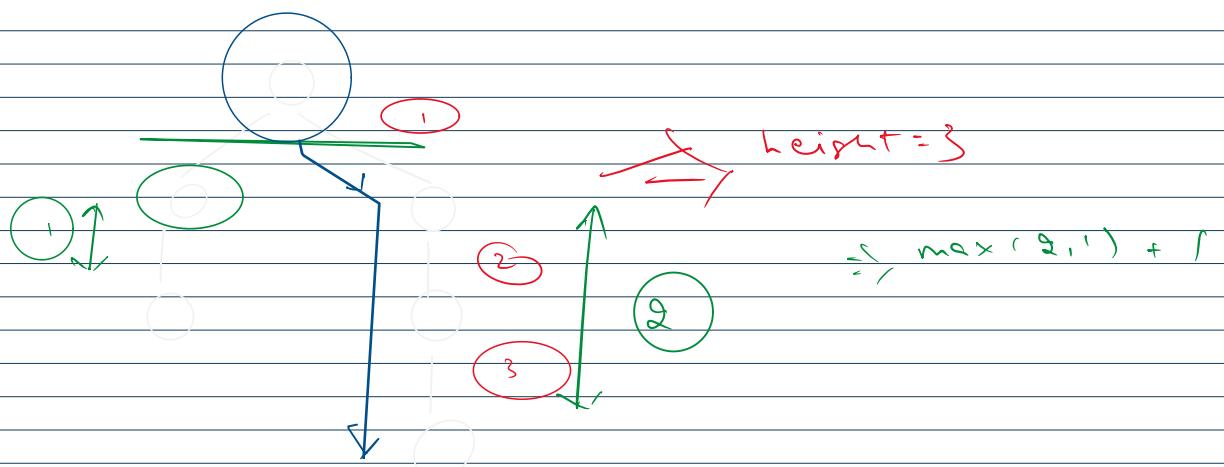


Diameter of a Binary Tree



$\Rightarrow 5 \Rightarrow \text{diameter}$

$\Rightarrow 6 \Rightarrow \text{diameter}$



\Rightarrow height = 2

\Rightarrow height = 0

